```
//////////////////////
//////  Tiles.h  //////
//////////////////////
template<uint8_t Ndim>
class tiles{
private:
    std::vector<std::vector<int>> tiles_;
public:
    tiles() {}
    void resizeTiles() { tiles_.resize(nTiles); }

    int nTiles;
    std::array<float,Ndim> tilesSize;
    std::array<std::vector<float>,Ndim> minMax;

    int getBin(float coord_, int dim_) const {
        int coord_Bin = (coord_ - minMax[dim_][0])/tilesSize[dim_];
        coord_Bin = std::min(coord_Bin,(int)(std::pow(nTiles,1.0/Ndim)-1));
        coord_Bin = std::max(coord_Bin,0);
        return coord_Bin;
    }

    int getGlobalBin(std::vector<float> coords) const {
        int globalBin = getBin(coords[0],0);
        int nTilesPerDim = std::pow(nTiles,1.0/Ndim);
      for(int i = 1; i != Ndim; ++i) {
            globalBin += nTilesPerDim*getBin(coords[i],i);
      }
      return globalBin;
    }

    int getGlobalBinByBin(std::vector<int> Bins) const {
        int globalBin = Bins[0];
      int nTilesPerDim = std::pow(nTiles,1.0/Ndim);
      for(int i = 1; i != Ndim; ++i) {
            globalBin += nTilesPerDim*Bins[i];
      }
      return globalBin;
    }

    void fill(std::vector<float> coords, int i) {
      tiles_[getGlobalBin(coords)].push_back(i);
    }

    void fill(std::vector<std::vector<float>> const& coordinates) {
        auto cellsSize = coordinates[0].size();
      for(int i = 0; i < cellsSize; ++i) {
            std::vector<float> bin_coords;
            for(int j = 0; j != Ndim; ++j) {
                bin_coords.push_back(coordinates[j][i]);
            }
            tiles_[getGlobalBin(bin_coords)].push_back(i);
        }
    }

    std::array<int,2*Ndim> searchBox(std::array<std::vector<float>,Ndim> minMax_){   // {{minX,maxX},{minY,ma
        std::array<int, 2*Ndim> minMaxBins;
      int j = 0;
      for(int i = 0; i != Ndim; ++i) {
            minMaxBins[j] = getBin(minMax_[i][0],i);
            minMaxBins[j+1] = getBin(minMax_[i][1],i);
            j += 2;
      }

        return minMaxBins;
    }

    void clear() {
        for(auto& t: tiles_) {
            t.clear();
```

```
        }
    }

    std::vector<int>& operator[](int globalBinId) {
        return tiles_[globalBinId];
    }
};


////////////////////////////
//////  clustering.h  //////
////////////////////////////
template <uint8_t Ndim>
class ClusteringAlgo{
public:
    int calculateNTiles(int pointPerBin) {
        int ntiles = points_.n/pointPerBin;
        try{
            if(ntiles == 0) {
                throw 100;
            }
        }
        catch(...) {
            std::cout << "pointPerBin is set too high for you number of points. You must lower it in the clus
        }
        return ntiles;
    }
};
////////////////////////////
//////  Binding module  //////
////////////////////////////
PYBIND11_MODULE(CLUEsteringCPP, m) {
    m.doc() = "Binding for CLUE";

    m.def("mainRun", &mainRun, "mainRun");
}
```